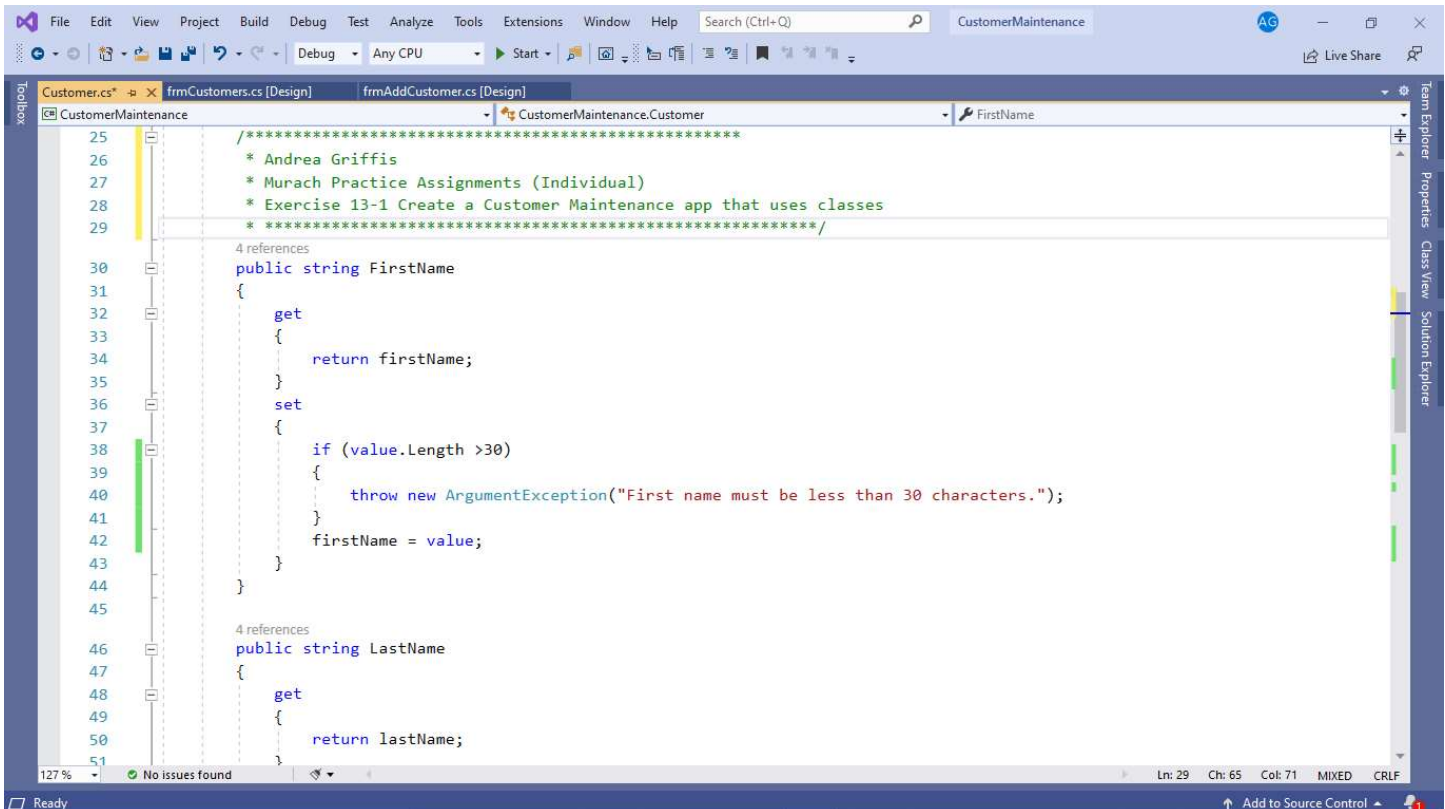


Assignment: Murach Practice Assignments (Individual)

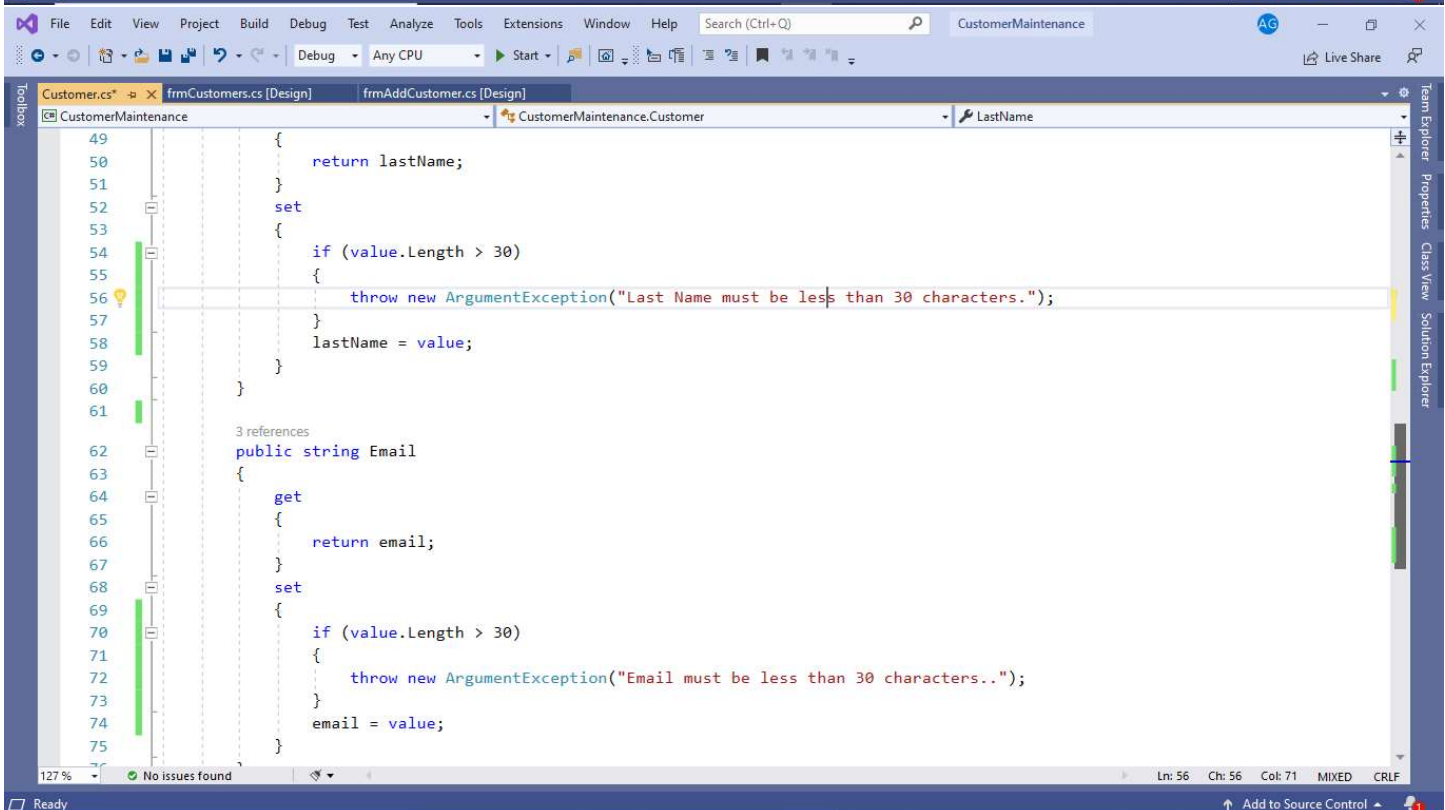
Exercise 13-1 Create a Customer Maintenance app that uses classes (#'s 1-16)

Open the project and add validation code to the customer class #1-4



The screenshot shows the Visual Studio IDE with the 'CustomerMaintenance' project open. The 'CustomerMaintenance.Customer' class is displayed in the 'frmAddCustomer.cs [Design]' view. The class has two public string properties: 'FirstName' and 'LastName'. The 'FirstName' property has a getter and a setter that throws an 'ArgumentException' if the value is longer than 30 characters. The 'LastName' property has a getter and a setter that also throws an 'ArgumentException' if the value is longer than 30 characters. The code is as follows:

```
25 /*****  
26  * Andrea Griffis  
27  * Murach Practice Assignments (Individual)  
28  * Exercise 13-1 Create a Customer Maintenance app that uses classes  
29  *****/  
30  
31 4 references  
32 public string FirstName  
33 {  
34     get  
35     {  
36         return firstName;  
37     }  
38     set  
39     {  
40         if (value.Length > 30)  
41         {  
42             throw new ArgumentException("First name must be less than 30 characters.");  
43         }  
44         firstName = value;  
45     }  
46  
47 4 references  
48 public string LastName  
49 {  
50     get  
51     {  
52         return lastName;  
53     }  
54     set  
55     {  
56         if (value.Length > 30)  
57         {  
58             throw new ArgumentException("Last Name must be less than 30 characters.");  
59         }  
60         lastName = value;  
61     }  
62  
63 3 references  
64 public string Email  
65 {  
66     get  
67     {  
68         return email;  
69     }  
70     set  
71     {  
72         if (value.Length > 30)  
73         {  
74             throw new ArgumentException("Email must be less than 30 characters..");  
75         }  
76         email = value;  
77     }  
78 }
```

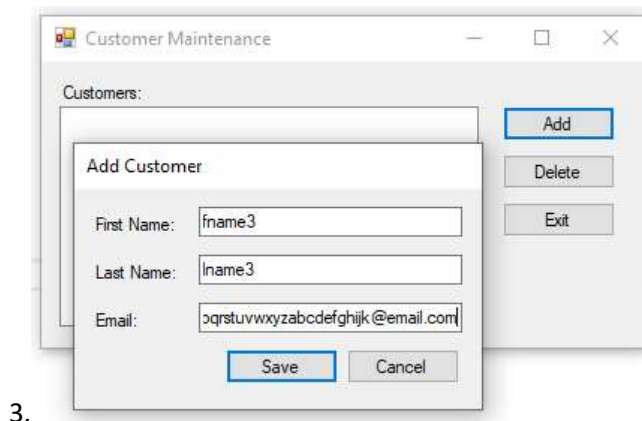
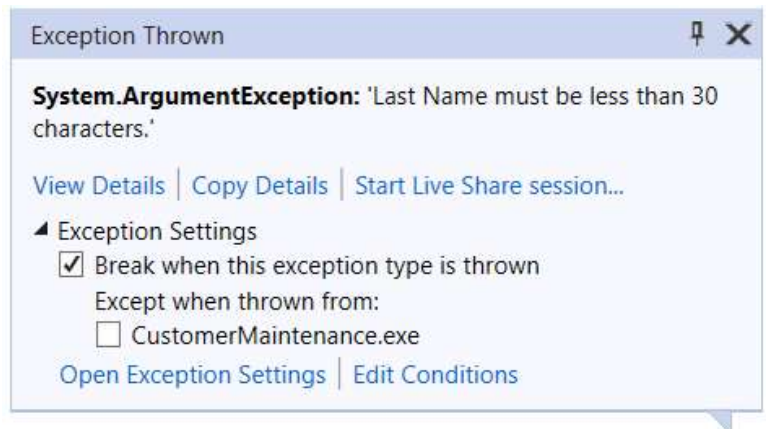
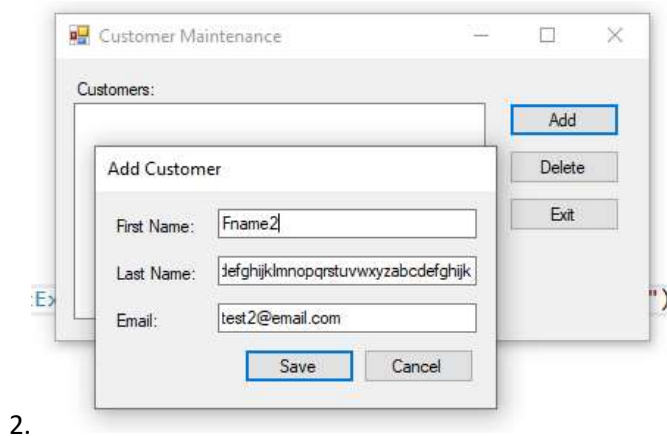
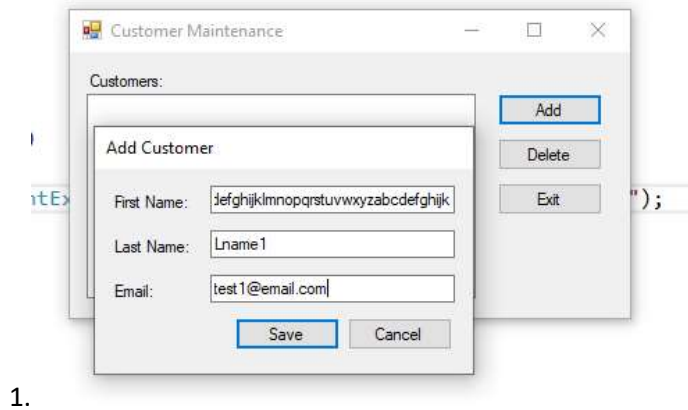


The screenshot shows the Visual Studio IDE with the 'CustomerMaintenance' project open. The 'CustomerMaintenance.Customer' class is displayed in the 'frmAddCustomer.cs [Design]' view. The class has three public string properties: 'FirstName', 'LastName', and 'Email'. The 'Email' property has a getter and a setter that throws an 'ArgumentException' if the value is longer than 30 characters. The code is as follows:

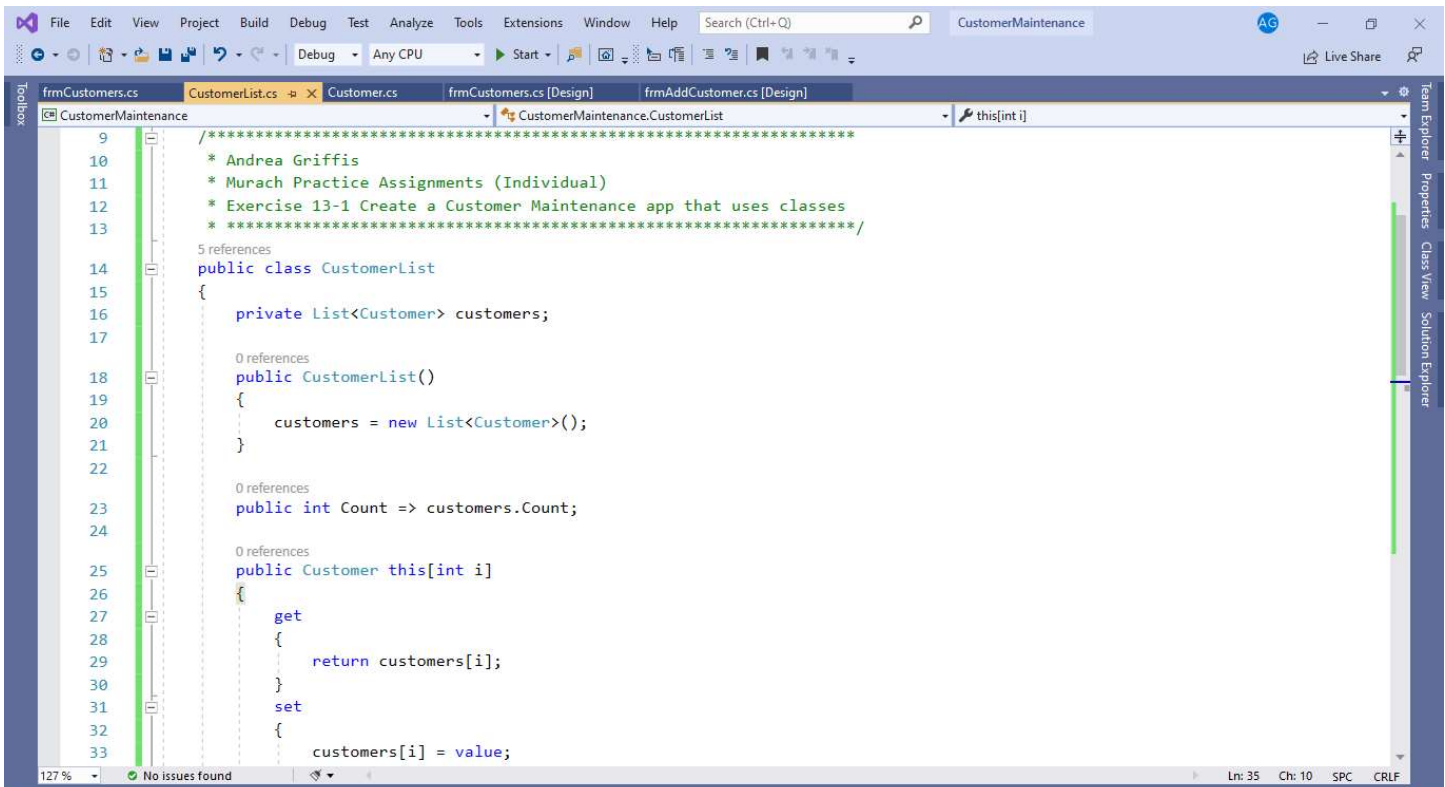
```
49 {  
50     return lastName;  
51 }  
52 set  
53 {  
54     if (value.Length > 30)  
55     {  
56         throw new ArgumentException("Last Name must be less than 30 characters.");  
57     }  
58     lastName = value;  
59 }  
60  
61  
62 3 references  
63 public string Email  
64 {  
65     get  
66     {  
67         return email;  
68     }  
69     set  
70     {  
71         if (value.Length > 30)  
72         {  
73             throw new ArgumentException("Email must be less than 30 characters..");  
74         }  
75         email = value;  
76     }  
77 }
```

Test the application for > 30 characters

- 1) 1st name>30 2) 2nd name >30 3) Email >30

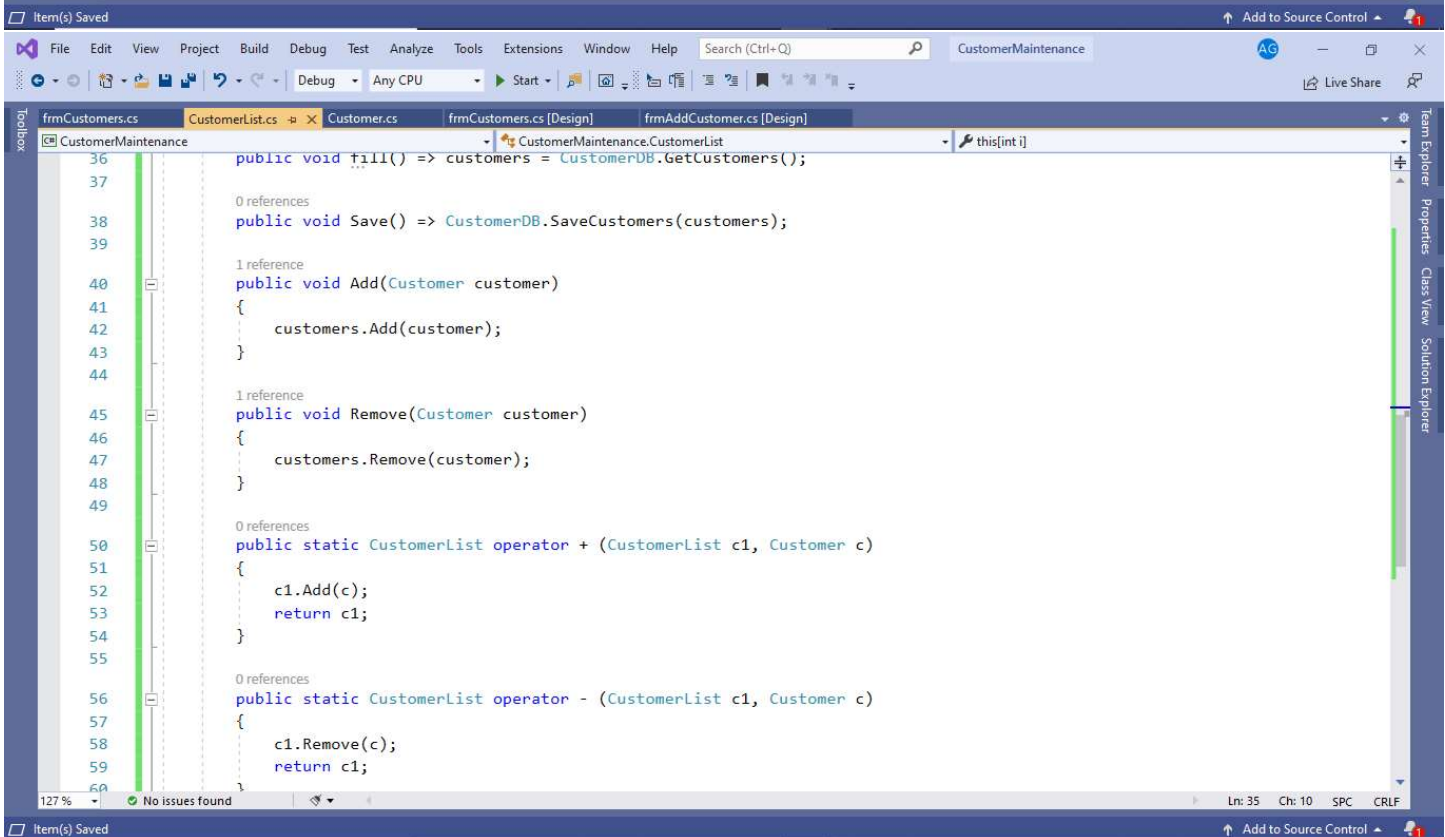


Add a CustomerList class #5-8



This screenshot shows the Visual Studio IDE with the `CustomerMaintenance.cs` file open. The `CustomerList` class is defined with a private `customers` list and several public methods. The code is as follows:

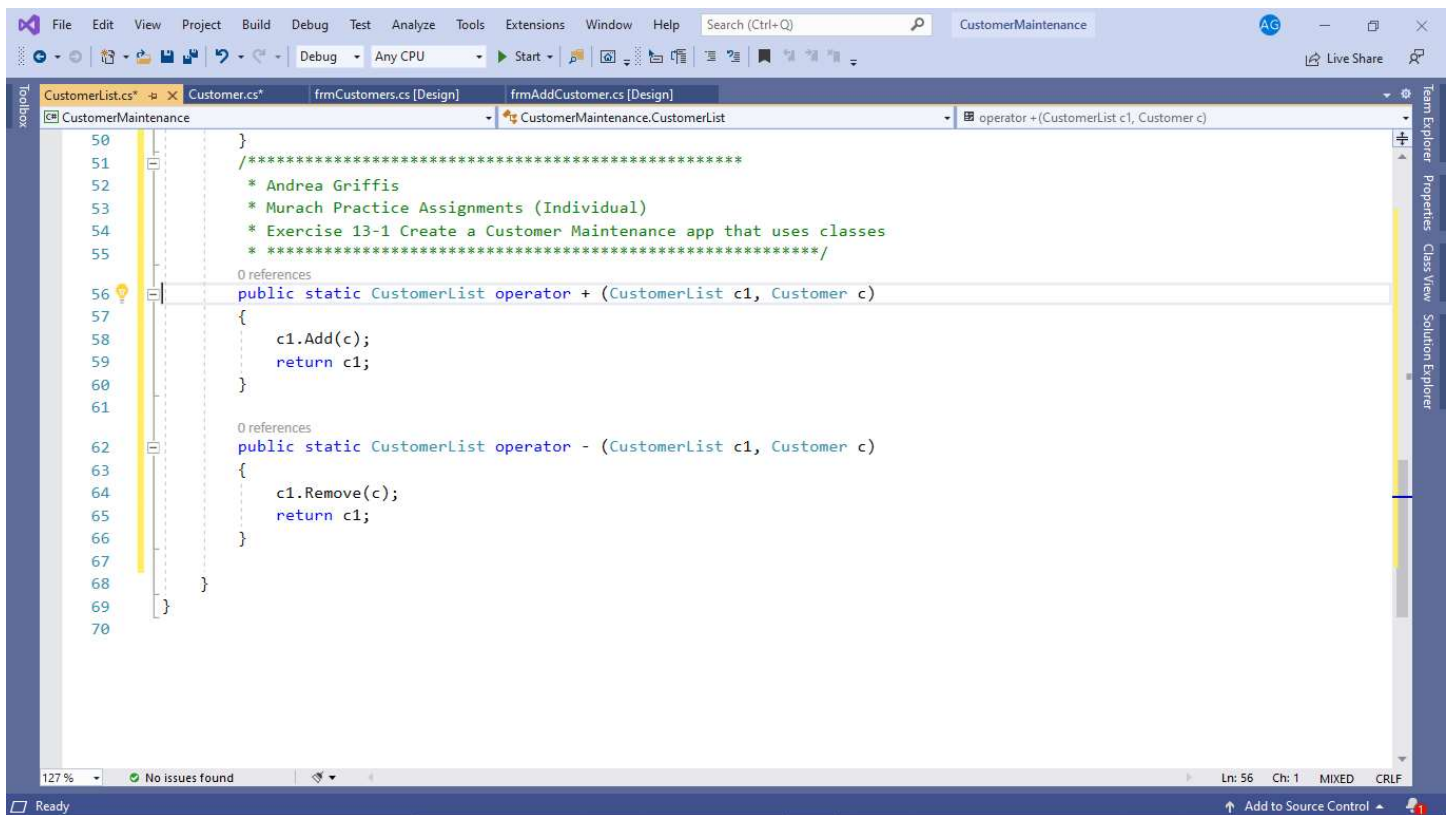
```
9  /*****  
10 * Andrea Griffis  
11 * Murach Practice Assignments (Individual)  
12 * Exercise 13-1 Create a Customer Maintenance app that uses classes  
13 *****/  
14 public class CustomerList  
15 {  
16     private List<Customer> customers;  
17  
18     0 references  
19     public CustomerList()  
20     {  
21         customers = new List<Customer>();  
22     }  
23  
24     0 references  
25     public int Count => customers.Count;  
26  
27     0 references  
28     public Customer this[int i]  
29     {  
30         get  
31         {  
32             return customers[i];  
33         }  
34         set  
35         {  
36             customers[i] = value;  
37         }  
38     }  
39 }
```



This screenshot shows the continuation of the `CustomerList` class implementation in `CustomerMaintenance.cs`. The code includes methods for filling the list, saving, adding, and removing customers, as well as static operators for addition and subtraction.

```
36 public void fill() => customers = CustomerDB.GetCustomers();  
37  
38 0 references  
39 public void Save() => CustomerDB.SaveCustomers(customers);  
40  
41 1 reference  
42 public void Add(Customer customer)  
43 {  
44     customers.Add(customer);  
45 }  
46  
47 1 reference  
48 public void Remove(Customer customer)  
49 {  
50     customers.Remove(customer);  
51 }  
52  
53 0 references  
54 public static CustomerList operator + (CustomerList c1, Customer c)  
55 {  
56     c1.Add(c);  
57     return c1;  
58 }  
59  
60 0 references  
61 public static CustomerList operator - (CustomerList c1, Customer c)  
62 {  
63     c1.Remove(c);  
64     return c1;  
65 }
```

Add overloaded operators to the CustomerList class #9-10

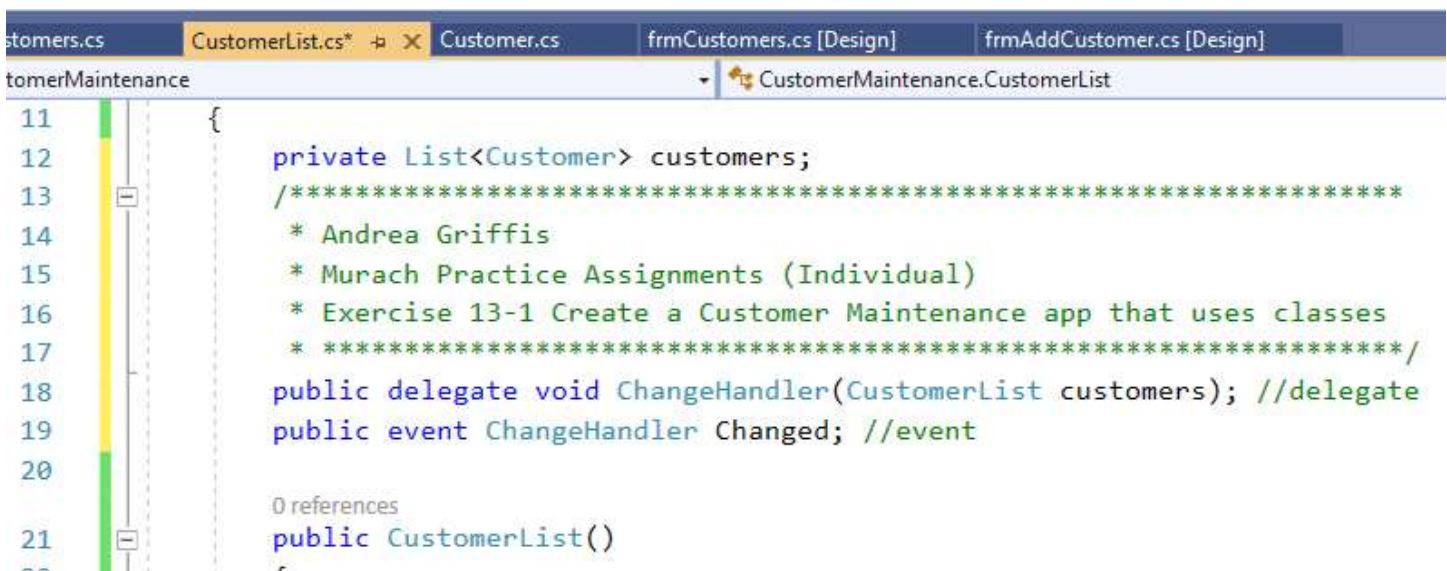


The screenshot shows the Visual Studio IDE with the 'CustomerMaintenance' project open. The 'CustomerList.cs' file is selected, and the 'operator + (CustomerList c1, Customer c)' method is being implemented. The code includes a comment block with the author's name and exercise details. The implementation for the '+' operator adds a customer to the list and returns the list. The implementation for the '-' operator removes a customer from the list and returns the list.

```
50 }
51 /*****
52  * Andrea Griffis
53  * Murach Practice Assignments (Individual)
54  * Exercise 13-1 Create a Customer Maintenance app that uses classes
55  * *****/
56 0 references
57 public static CustomerList operator + (CustomerList c1, Customer c)
58 {
59     c1.Add(c);
60     return c1;
61 }
62 0 references
63 public static CustomerList operator - (CustomerList c1, Customer c)
64 {
65     c1.Remove(c);
66     return c1;
67 }
68 }
69
70
```

Add a delegate and an event to the CustomerList class #12-16

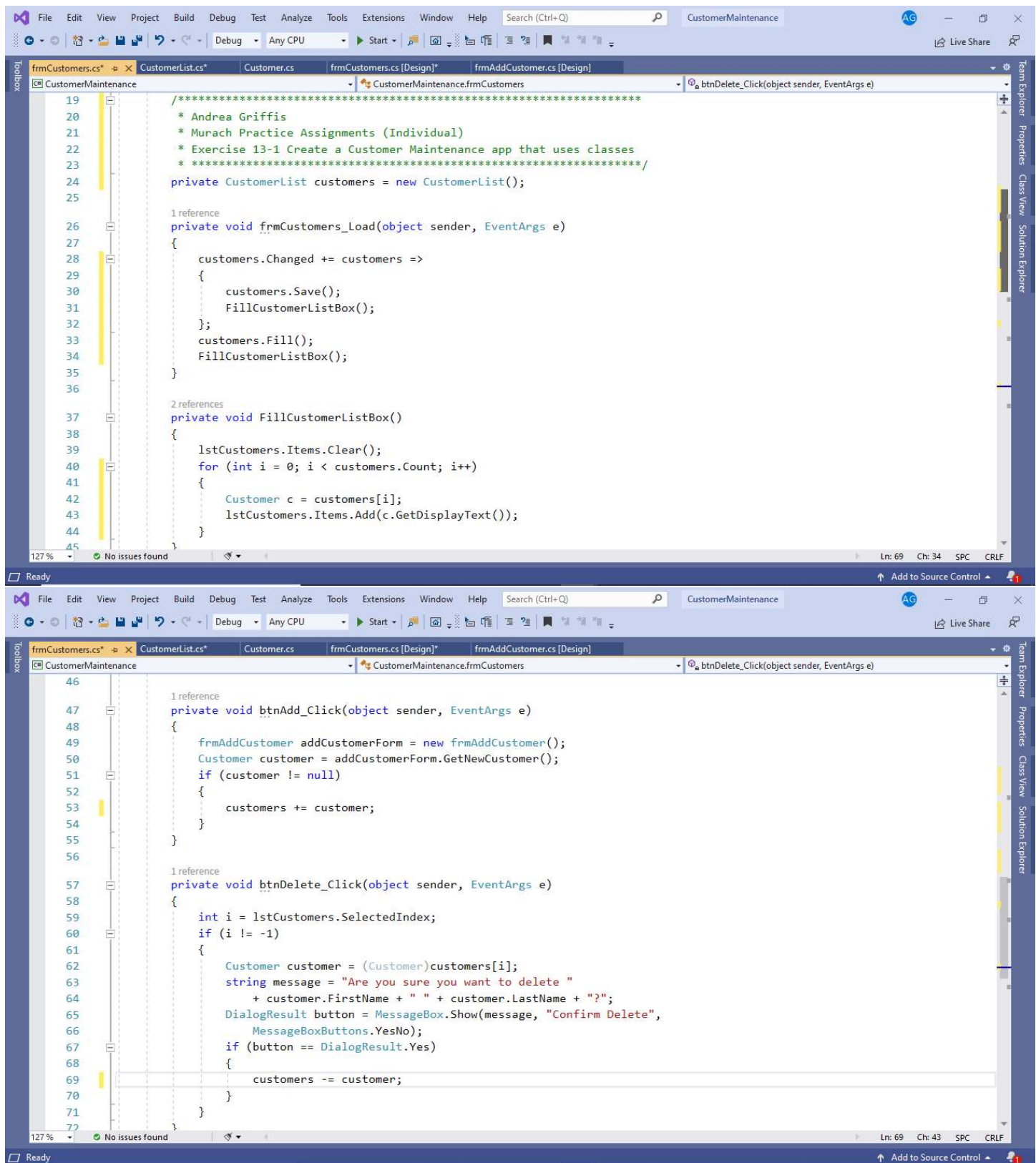
(11 & 12)



The screenshot shows the Visual Studio IDE with the 'CustomerMaintenance' project open. The 'CustomerList.cs' file is selected, and the 'CustomerList' class is being implemented. The code includes a comment block with the author's name and exercise details. The implementation for the '+' operator adds a customer to the list and returns the list. The implementation for the '-' operator removes a customer from the list and returns the list.

```
11 {
12     private List<Customer> customers;
13     /*****
14      * Andrea Griffis
15      * Murach Practice Assignments (Individual)
16      * Exercise 13-1 Create a Customer Maintenance app that uses classes
17      * *****/
18     public delegate void ChangeHandler(CustomerList customers); //delegate
19     public event ChangeHandler Changed; //event
20
21     0 references
22     public CustomerList()
23     {
24     }
```


(13-15)



The image displays two screenshots of a Visual Studio IDE window, showing the implementation of a Customer Maintenance application. The top screenshot shows the `frmCustomers_Load` method and the `FillCustomerListBox` method. The bottom screenshot shows the `btnAdd_Click` and `btnDelete_Click` methods.

Top Screenshot Code:

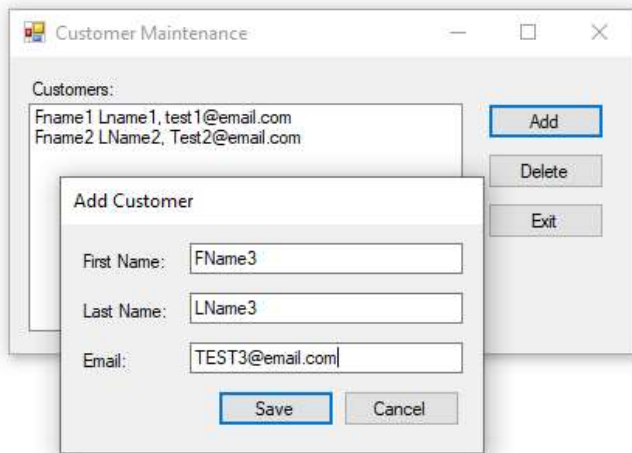
```
19 /*****  
20  * Andrea Griffis  
21  * Murach Practice Assignments (Individual)  
22  * Exercise 13-1 Create a Customer Maintenance app that uses classes  
23  * *****/  
24 private CustomerList customers = new CustomerList();  
25  
26 1 reference  
27 private void frmCustomers_Load(object sender, EventArgs e)  
28 {  
29     customers.Changed += customers =>  
30     {  
31         customers.Save();  
32         FillCustomerListBox();  
33     };  
34     customers.Fill();  
35     FillCustomerListBox();  
36 }  
37 2 references  
38 private void FillCustomerListBox()  
39 {  
40     lstCustomers.Items.Clear();  
41     for (int i = 0; i < customers.Count; i++)  
42     {  
43         Customer c = customers[i];  
44         lstCustomers.Items.Add(c.GetDisplayText());  
45     }  
46 }
```

Bottom Screenshot Code:

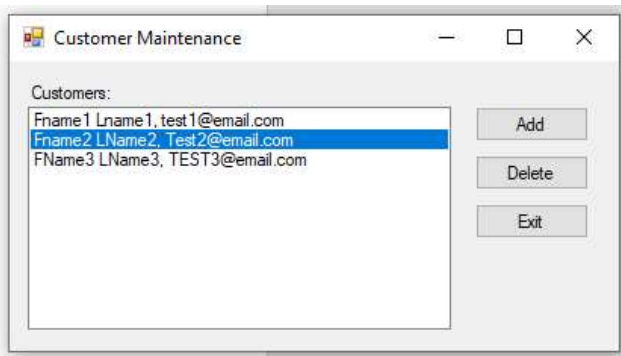
```
47 1 reference  
48 private void btnAdd_Click(object sender, EventArgs e)  
49 {  
50     frmAddCustomer addCustomerForm = new frmAddCustomer();  
51     Customer customer = addCustomerForm.GetNewCustomer();  
52     if (customer != null)  
53     {  
54         customers += customer;  
55     }  
56 }  
57 1 reference  
58 private void btnDelete_Click(object sender, EventArgs e)  
59 {  
60     int i = lstCustomers.SelectedIndex;  
61     if (i != -1)  
62     {  
63         Customer customer = (Customer)customers[i];  
64         string message = "Are you sure you want to delete "  
65         + customer.FirstName + " " + customer.LastName + "?";  
66         DialogResult button = MessageBox.Show(message, "Confirm Delete",  
67         MessageBoxButtons.YesNo);  
68         if (button == DialogResult.Yes)  
69         {  
70             customers -= customer;  
71         }  
72     }  
73 }
```

#16 Test the application

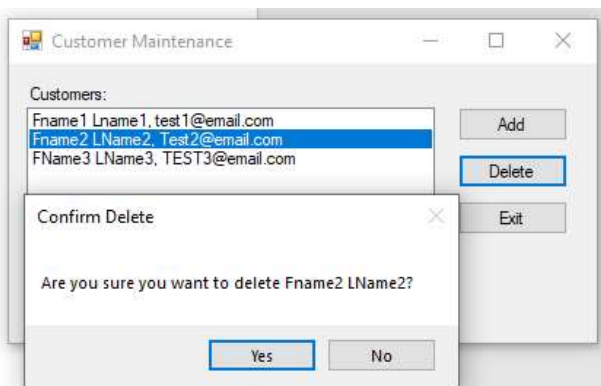
- 1) Add customers 2) Choose Customer to delete
- 3) Confirm Delete 4) Completed Delete



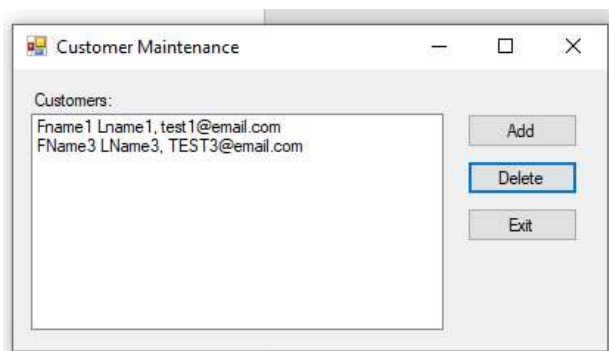
1.



2.



3.



4.